SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE, VOL. 9, 379–399 (1997)

Opinion

Software Maintenance and Global Competitiveness

DON O'NEILL*
9305 Kobe Way, Gaithersburg MD 20879–2101, U.S.A.

SUMMARY

Effective software maintenance is essential to achieving global software competitiveness. Organizations' increasing dependency on software and the shortage of software skills combine to require an increase in the span of responsibility within the software maintenance function.

The span of responsibility determines the cost of software ownership since it is the amount of software depended on by the enterprise divided by the number of people who maintain the software. Managers of software maintenance can improve the span of responsibility by

- **establishing and using organizational resources in five manners to sustain operational support for existing information systems,**
- **applying resources to strengthen four views of software maintenance, from individual to national,**
- **creating and applying action to eliminate the 26 main shortfalls in doing software maintenance, and**
- **reducing the seven main risks to information system quality.**

This paper provides a call for action to improve software maintenance in order to strengthen enterprise and national global competitiveness. © 1997 John Wiley & Sons, Ltd.

J. Softw. Maint.: Res. Pract., **9**, 379–399 (1997)

No. of Figures: 8. No. of Tables: 1. No. of References: 23.

KEY WORDS: span of responsibility; maintenance shortfalls; defect-type frequency; sustaining operational support; commodity software; value-add software

1. CONTEXT

Over the past 40 years, a trillion lines of code have been written. Software usage is increasing with more industries becoming dependent on software, and dependency within each industry deepening. These applications were produced for industries of all kinds including telecommunications, transportation, financial, manufacturing, medical systems, and utilities and energy.

While these systems were being developed, very few managers (the main exception being in telecommunications) expected the systems to be operating and maintained for a

* Correspondence to: Don O'Neill, Independent Consultant, 9305 Kobe Way, Gaithersburg MD 20879–2101, USA. E-mail: ONeillDon@aol.com

CCC 1040–550X/97/060379–21$17.50

Received 20 August 1997
Revised 12 September 1997

quarter of a century or more. The systems were not produced with maintenance in mind. They were not engineered and designed for long-term maintenance. Consequently, the legacy systems that provide the backbone of the nation's critical industries do so without traceability from the code to the requirements, without compliance to design and programming standards, with dead code, with extra code, and with commented-out code. Some of this has been due to a lack of good practice, some to neglect. This is the world's and the USA's legacy of software needing the attention of software engineering.

The result is that software users now dedicate about 70 per cent of their software workforce to the maintenance of their products (Nosek and Palvia, 1990). At the same time, software skills are in short supply. Consequently, the number of lines of code maintained or to be maintained per full-time-equivalent person doing software maintenance is a critically important metric. This metric is called, for short hereafter, the *span of responsibility*. A historical note about this metric may be helpful here. In the past, the term 'span of control' (number of persons supervised who work together) has been associated with the character of the structure of an organization (Drucker, 1974, pp. 412–414). A tall organizational structure with many levels is characterized by a small span of control, such as less than five people. A flat structure with few levels is characterized by a large span of control, such as more than 15 people.

The term 'span of responsibility' is analogous to the term 'span of control' but reflects changing conditions and focus. Today with the de-emphasis on organizational structure and increased emphasis on personnel empowerment, the span of control is now less important than the span of responsibility. This is because the span of responsibility puts the focus on the need for personnel accomplishment rather than on personnel head count. However, because of the analogy between the two terms, some people (including the author, because of Drucker's emphasis (1974, p. 602)) would like to redefine 'span of control' to be the 'span of responsibility'. In software maintenance, the author observes that the span of responsibility currently ranges from about 35 000 lines of code to about 250 000 lines of code maintained or to be maintained.

2. BASIC CONCEPTS

What is software? The view that software facilitates the harmonious co-operation between people and machines is difficult to improve upon as a starting point for the definition of software. It certainly beats those descriptions that point out that software is weightless and invisible, the non-hardware parts of a computer-implemented system (Rosen, 1993, p. 1214).

What is software engineering? The definition of software engineering evolved as a part of information technology (IT) for nearly 25 years before it included software maintenance (O'Neill, 1989). Three partial definitions of software engineering are these:

1. The term software engineering first gained wide attention at the 'Software Engineering' conference in October 1968 in Garmish in Germany sponsored by the NATO Science Committee (Buxton, Naur and Randell, 1976, p. 5). 'The phrase "software engineering" was deliberately chosen to be provocative, in implying the need for

software manufacture to be used on the types of theoretical foundations and practical disciplines that are traditional in the established branches of engineering.'

2. In the *Scientific American*, Gibbs (1994) provided a definition of software engineering as: 'The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.'
3. The *Encyclopedia of Computer Science* edited by Ralston and Reilly (Humphrey, 1993, p. 1218) defines software engineering as '. . .that form of engineering that applies the principles of computer science and mathematics to achieving cost-effective solutions to software problems.'

What is expected of software products? Simply stated, a software product should provide the right answers on time, every time. The current practices of software development and maintenance yield software with three to four defects per thousand lines of source code (KLOC). When management support and 'black belt' programs operate to ensure enterprise quality achievement, leading corporations have achieved software with 0·3 to 0·4 defects per KLOC. Higher quality software than that is proving elusive.

What is expected of software processes? The software process should be dependable with respect to cost, schedule and quality. Dependability is what is sought by the SEI's CMM (Software Engineering Institute's Capability Maturity Model) when it associates predictability with level 3 maturity, a level achieved by less than 15·2% of those sites assessed (SEI, 1997).

What is expected of the post-deployment process? Once developed and fielded, the software product requires maintenance to sustain and support its operation. This process following deployment should be capable of recreating the software itself as well as recreating the tools used to create it. With this capability, the enterprise is able to evolve and change the software product and to port the software product and its tools onto modern platforms.

What is software maintenance? Software maintenance involves changing existing software. The IEEE (1993, p. 4) definition defines software maintenance this way: 'Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.' In the short term, software is changed to make it good enough. Longer term, it is changed to make it better. While a narrow view of what is maintenance is that maintenance involves corrective, perfective and adaptive operations on existing systems, implemented at least in part with software—legacy code, is a good example—a broader view is that maintenance involves the full range of a sustaining operational support needed to ensure long term confident ownership of software products. In managing software maintenance operations, it is useful to distinguish between corrective and improvement metrics (Humphrey, 1997).

3. SUSTAINING OPERATIONAL SUPPORT

3.1. Goals

It is necessary to understand what the customer wants most in supportability and to align the capability of the organization to provide it. Certainly, it is necessary to sustain

continuous operation in the software systems supported. When there is a deviation, it is necessary to return the software system to normal operation in a timely fashion. To accomplish this dependably, it is necessary to increase reliance on an organization's process capability while decreasing dependence on specific individuals. Setting supportability objectives for each software system supported and measuring performance against those objectives facilitate the shift from people to process.

3.2. Commitment

The enterprise demonstrates its commitment by extending the organization's infrastructure to include sustaining operational support. This infrastructure includes preparing and issuing an organizational supportability policy and procedure, training personnel in the knowledge and skills needed for the expert practice of operational support, and budgeting and allocating the resources to carry out the policy.

3.3. Practice

The expert practice of operational support begins by forging a shared vision among development, acquisition and operations, and using communities on the expectations for supportability. It includes producing and sustaining a domain-specific reference architecture and its supporting documentation, including the rationale and test materials. Well-defined, fine-grained engineering and management processes for supportability are produced in accordance with the organization's policy and procedure. These include the capability to convene rapidly any needed emergency-response teams.

3.4. Measurement

The major indicators of supportability readiness and practice first have to be identified and thereafter tracked consistently. Some of these indicators include software product size, span of responsibility, cyclomatic and essential complexity, computer resource loading history, change history, defect history, history of requirements conformance, system availability, and system reliability.

3.5. Oversight

The actual practice of supportability is verified through customer satisfaction surveys and complaint logs. Periodic reviews and inspections of the domain-specific reference architecture and supporting documentation are conducted. The degree of satisfactory completion of exit criteria, and the predictability of each well-defined, fine-grained engineering and management process are monitored.

4. MULTIPLE VIEWS OF SOFTWARE MAINTENANCE

4.1. Individual view

Software maintenance can be viewed and managed at many levels. At an individual level, a programmer with software maintenance responsibility is likely to be at work on a Saturday morning charged with detecting and correcting a critical defect impacting customer operations. Like a detective, the programmer pores over the program looking for clues. There could be a clue if the code is traceable to the design, specification and requirements documents; but rarely is it traceable. In fact, these life cycle documents are not always produced even as part of a development project, and even if produced may not be maintained after initial deployment. There could be a clue in the comments embedded in the code; but usually there are few comments, and those that are there cannot be trusted. As the programmer steps through sequences of code looking for clues, the hard fact of software maintenance practice is apparent. The code does what the code does. Without good traceability to and from trusted documentation and code comments, software maintenance becomes a crap shoot.

4.2. Project

On the project, the best technical practices needed for software maintenance can be drawn from the SEI's CMM covering software requirements management software configuration management, and software quality assurance from level 2, and software product engineering and peer reviews from level 3 (Humphrey, 1989). In addition, software regression testing, complexity analysis (for example, McCabe and Watson (1994)) and software patching are needed.

4.3. Enterprise

The enterprise responsibility is to establish and sustain the operational support infrastructure and to populate it with trained and competent people. The demand for qualified software personnel is increasingly outdistancing the supply. At the same time enterprise dependency on software is increasing. More is expected of less. Therefore, an enterprise focus on managing the span of responsibility is relevant to sustaining current operations and to competing for the future.

The enterprise maintaining legacy code and seeking to be competitive needs to sustain an ever increasing volume of code with a level head count. The span of responsibility must increase. Yet the span of responsibility is typically not measured and managed in practice. As noted earlier, the author observes a wide variation currently in the span of responsibility from enterprise to enterprise from less than 35 000 lines per person to more than 250 000.

Near term, the enterprise can increase its span of responsibility by efficient processes that minimize defect leakage into field operations. Long term the enterprise must utilize technology to increase the bandwidth of application knowledge with which a single

individual is charged. For example, if traceability is provided from the code to the design, specification and requirements, and these documents are hypertext-linked so that a code fragment can be viewed with the applicable design, specification and requirements, the intellectual bandwidth of the maintenance programmer can be greatly expanded. A systematic software product assessment may provide the improvement agenda needed to raise the span of responsibility (Salisbury, 1997).

4.4. Nation

Obtaining prosperity for its citizens is a common goal for a nation. Being globally competitive is becoming the responsibility of the enterprise for it to prosper within a nation. Yet it is the nation that sets the research and development priorities and supplies the resources to carry them out. Note that in the USA, Federally funded research and development has decreased from 2·8% of the GDP (gross domestic product) to 2·2% in the past ten years. This is behind Japan which has announced plans to double its government-funded research and development. Capers Jones reports that Japanese software quality is measured at 0·32 defects per function point while the rate in the USA is 0·75 defects per function point with the rate in Canada at 0·64 (Jones, 1997).

5. INDICATORS OF EFFECTIVE MAINTENANCE

5.1. National Software Council effort

In order to increase awareness within society and understanding among policy makers about the value of software in achieving national prosperity, the National Software Council (NSC) is sponsoring a program to improve the understanding of the value of software to the national economy and global competitiveness (O'Neill, 1997a). The NSC has identified the major indicators of global software competitiveness. Figure 1 diagrams the general

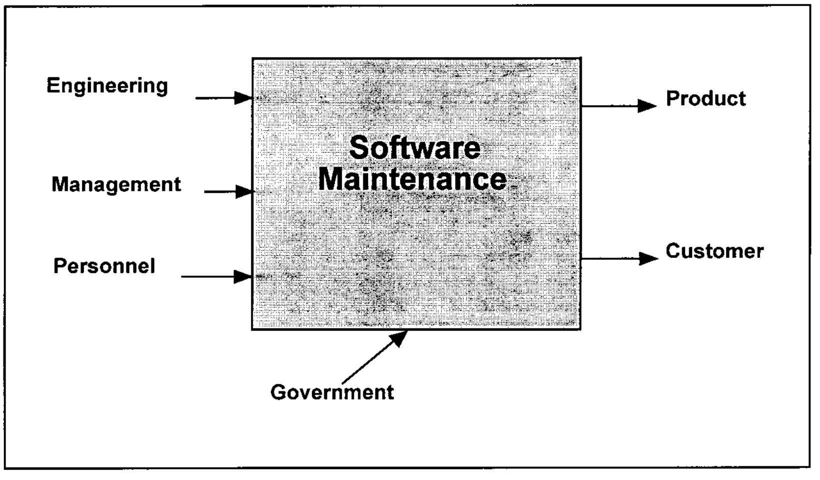

Figure 1. Summary of the categories of indicators relevant to software maintenance

relationships among categories of the relevant major indicators applicable to software maintenance.

The indicators useful for assessing the effectiveness of software maintenance are grouped below under engineering, management, personnel, product, customer and government. Each indicator is described, and a comment on the state of each indicator is provided. In conducting assessments of the global software competitiveness of an enterprise, each indicator is scored on a judgement basis on a scale from one to five for readiness R. Each indicator is also scored in the same way and on the same scale for importance I, but with the restriction that I cannot be less than R. From this, a shortfall metric S can be calculated from the formula

$$S = (I - R) \div I \tag{1}$$

Bar graphs of aggregate estimates of the shortfall metric in the USA in 1997 are shown below for each maintenance type. High bars indicate major shortfall, and the closer the bar height is to zero, the smaller is the estimated shortfall. The source for all of the estimates is the National Software Council (O'Neill, 1997a).

5.2. Engineering practices

5.2.1. Measurement of critical aspects

To achieve effective maintenance, the computer-using enterprise understands that it must measure the critical aspects of current practice (see Figure 2). Even though Lord

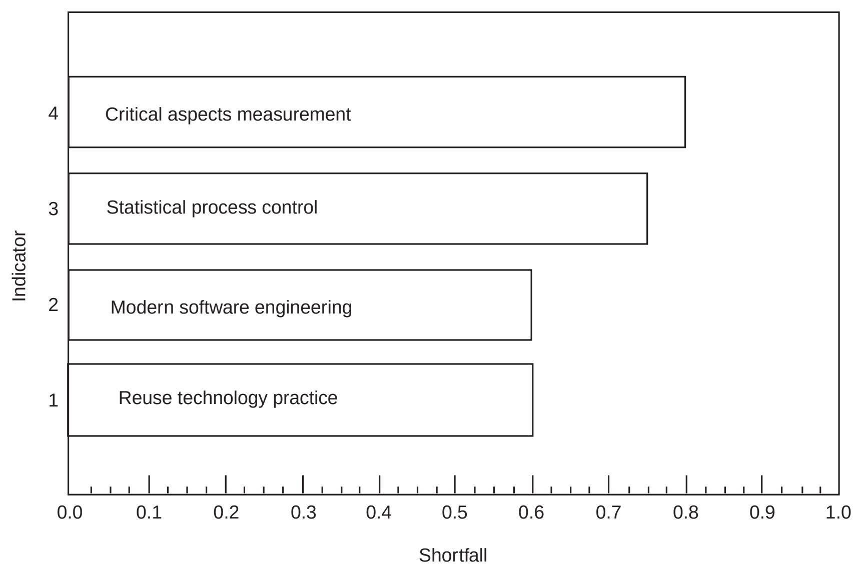

Figure 2. Estimated shortfalls in engineering practices

Kelvin (Thompson, 1992) cited how lack of measurement results in meagre and unsatisfactory knowledge, the practice of measurement in software is immature. Capable, consistent measurement of the critical aspects of software processes and its results are associated with SEI CMM level 4, achieved by only 2·1% of those assessed (SEI, 1997).

5.2.2. Modern software engineering

To achieve effective maintenance, the enterprise understands that it must apply modern software engineering practice in the production and maintenance of its software products (see Figure 2). With capable committed management, applying modern software engineering can yield high quality software products from processes that are predictable with respect to cost, schedule and quality. Capable modern software engineering is associated with SEI CMM level 3, achieved by only 15·2% of those assessed (SEI, 1997).

5.2.3. Reuse technology practice

To achieve effective maintenance, the enterprise understands that it must practise software reuse in order to increase the span of responsibility (see Figure 2). The enterprise with a capable reuse technology practice is able to roll out modified products and features on short and predictable cycle times (Schach, 1994). Reuse practice is dependent on capable domain architecture practice, and benefits from capable requirements determination and specification practice. These practices are not commonly in use currently.

5.2.4. Statistical process control

To achieve effective maintenance, the enterprise understands that it must apply statistical process control in the management and engineering of its software operations (see Figure 2). Capable statistical process control is obtained at SEI CMM level 4, achieved by less than 2·1% of those assessed (SEI, 1997).

5.3. Management practices

5.3.1. Commodity view

To achieve effective maintenance, the enterprise classifies its software along a spectrum ranging from commodity to critical resource. Figure 3 gives the typical picture. Unenlightened enterprises continue to view programming as a commodity while, in reality, programmers are in very short supply, and demand is increasing faster than supply. The Information Technology Association of America (ITAA, 1997) reports 190 000 unfilled, open requisitions for information technology personnel.

5.3.2. Competitive wage structure

To achieve effective maintenance, the enterprise maintains a competitive wage structure for its software personnel and understands the trade-offs of outsourcing both domestic and offshore (see Figure 3). Foreign wage structures may be dramatically lower than

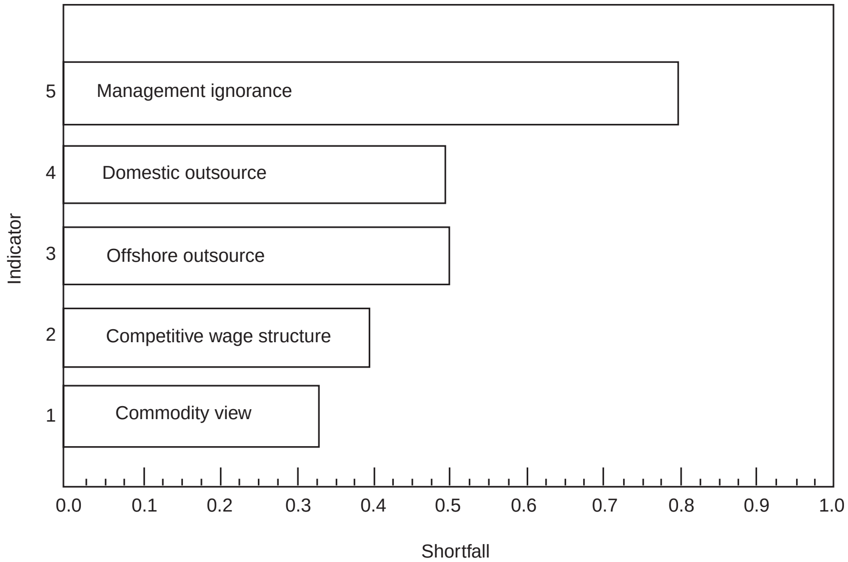

Figure 3. Estimated shortfalls in management practices

domestic. Where well trained programmers are in good supply, as in India, these competitive wage structures are an attraction to offshore outsourcing.

5.3.3. Domestic outsourcing

To achieve effective maintenance, the enterprise considers engaging in some level of domestic outsourcing in order to obtain and sustain this capability (see Figure 3). The enterprise that understands what its customers need most aligns its best capabilities and core competencies to provide that. Operations that fall outside of these core competencies are being outsourced. Outsource vendor demand is growing substantially.

5.3.4. Management ignorance

To achieve effective maintenance, the enterprise understands that it must select capable software managers who are boundary spanners among disciplines, including management, technology, process and application domain (see Figure 3). Neglect and malpractice on software projects occur regularly, as 61·5% of the organizations assessed remain at SEI CMM level 1, below the level of management commitment (SEI, 1997). As Humphrey (1989) has observed, competent, effective managers are wanted by everyone but are in very short supply, with the practical result that the enterprise's current managers are as good as are available.

5.3.5. Offshore outsourcing

To achieve effective maintenance, the enterprise considers engaging in some level of offshore outsourcing, such as of routine or corrective maintenance, in order to obtain and

sustain this capability (see Figure 3). The shortage of skilled programmers, competitive wage scale differentials, and the enterprise retrenchment into core competencies are fueling offshore outsourcing. Where the enterprise has a capable software requirements determination, specification practice and configuration management, the offshore outsource can be highly effective (Kumar, Das and Netaji, 1996). Rigorous requirements determination and specification practice are not well developed and used currently.

5.4. Personnel resources

5.4.1. Employee morale

To achieve effective maintenance, the enterprise measures the employee morale of software personnel and establishes a policy with upper and lower control limits for its management (see Figure 4). For an increasing number of enterprises, software is the enterprise's most precious resource, and analysts and programmers are the enterprise's most critical skills. Employee morale for analysts and programmers is coupled to motivation factors and the technical challenges faced on the projects—the higher the difficulty, the higher the morale.

5.4.2. Key employee status

To achieve effective maintenance, the enterprise provides key employee status and special benefits and compensation to strategically essential software managers and programmers (see Figure 4). The enterprise is permitted to identify key employees considered critically essential to current operations, future prospects and competitiveness. Key

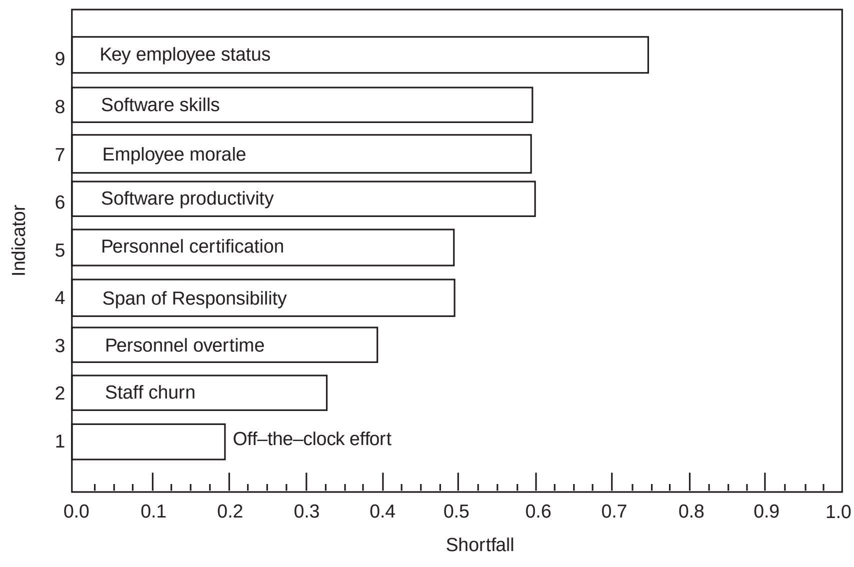

Figure 4. Estimated shortfalls from personnel practices

employee status brings with it special benefits, such as stock option retirement accounts taxed at capital gains rates. Historically, key employee status has been only infrequently assigned to software personnel. A 1997 Price Waterhouse study ranked retaining key software employees as the eighth most pressing concern of managers, up from 14th a year ago (Price Waterhouse, 1997).

5.4.3. Off-the-clock effort

To achieve effective maintenance, the enterprise measures the off-the-clock (i.e., unpaid 'voluntary' overtime) effort of its software maintenance personnel, and establishes a policy with upper and lower control limits for its management (see Figure 4). With the demand for programmers outstripping supply, with the fierce competition for new products and features to protect and extend market share, and with the falling margins caused by increased competition, the enterprise places demands and expectations on programmers that increase the practice of off-the-clock effort. Supplying employees with company lap top computers, at-home work stations and encouraging telecommuting further this trend.

5.4.4. Personnel certification

To achieve effective maintenance, the enterprise establishes and applies the training and experience criteria for each software skill type and level of achievement (see Figure 4). Programmer certification programs have existed for decades, such as the one by the Institute for the Certification of Computer Professionals, but have received little support from the employers of programmers. There is not an industry consensus on the knowledge, skills and behaviours in the design, management and operation of software systems. The professional and ethical foundations of software engineering are immature and require upgrading.

5.4.5. Personnel overtime

To achieve effective maintenance, the enterprise measures the on-the-clock overtime of its software personnel and establishes a policy with upper and lower control limits for its management (see Figure 4). The baseline work-week for maintenance programmers on projects with deadlines is often 60 to 75 hours. In the USA, analysts and programmers are usually exempt from mandatory overtime pay and commonly do not receive pay for overtime worked.

5.4.6. Software productivity

To achieve effective maintenance, the enterprise measures the productivity of its software personnel and understands the upper and lower control limits for tasks of various types and levels of difficulty (see Figure 4). With demand exceeding supply, project and programmer productivity are the focus of improvement. Productivity measurement depends on the persistent measurement of size (as, for example, via function points or lines of code) and effort (as, for example, via labour hours) organized by software maintenance life cycle activity (such as requirements, specification, design, code, test and operations)

(Chapin, 1988). At present, few enterprises measure size, effort and productivity in software work on a regular consistent basis.

5.4.7. Software skills

To achieve effective maintenance, the enterprise understands the software skills it needs and the software skills it possesses (see Figure 4). With industry software skills in short supply, enterprise personnel recruiting programs are unable to staff current projects. Openings for programming positions go unfilled. As noted before, the ITAA report that 190 000 open requisitions for information technology positions are unfilled.

5.4.8. Span of responsibility

To achieve effective maintenance, the enterprise measures the total number of lines of code in the systems which its product lines and operations depend upon, and the total number of personnel involved in the systems' maintenance (see Figure 4). The enterprise today is under pressure to support an ever increasing volume of code with a level head count. To do this, the span of responsibility must increase. In practice, the span of responsibility is rarely measured and managed.

5.4.9. Staff churn

To achieve effective maintenance, the enterprise understands the increasing risk of staff churn, its impact on current operations, and the personnel practices needed to counter it (see Figure 4). As the pull of new challenges and increased financial rewards are combined with the push of low employee morale, excessive overtime and off-the-clock time, the rate of programming staff turnover increases. The result is increased costs for hiring and training, impact to current commitments, and loss of knowledge and data to the competition. In high-turnover skill areas, employees may be required to sign non-competition covenants.

5.5. Software product

5.5.1. Conformance to requirements

To achieve effective maintenance, the enterprise understands that it must deliver software products that conform to the requirements of the customer and the industry. Figure 5 summarizes the shortfall. Conformance to requirements is a long-term indicator of software quality and is strongly institutionalized in the culture of software engineering.

5.5.2. Defect-free product

The enterprise understands that it must operate and deliver defect-free software products (see Figure 5). As noted previously, industry software practice in the USA produces three to four defects per thousand lines of code in operation. The total quality management goal for defect-free software is usually less than one defect per ten thousand lines of code. Times ten improvement programs are generally successful in moving to four defects

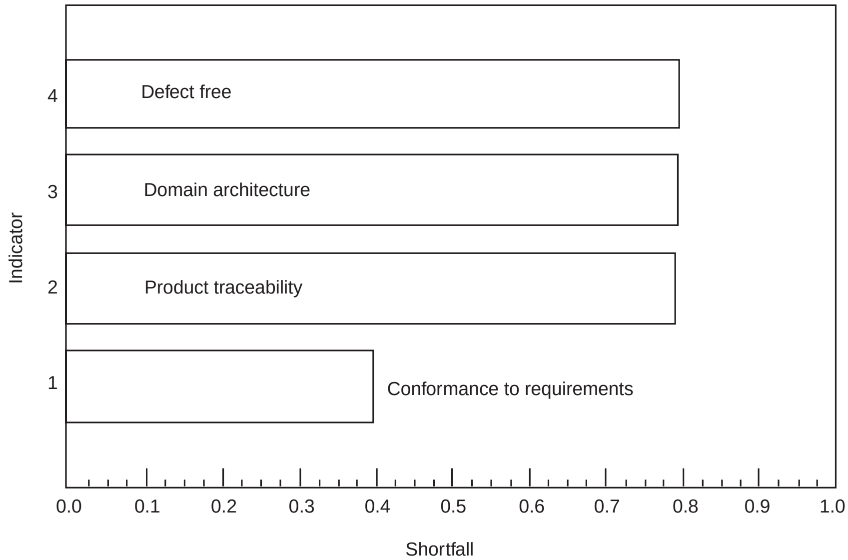

Figure 5. Estimated shortfalls from software product characteristics

per ten thousand lines of code by ‘plucking low hanging fruit’. Successive times ten improvement programs are occasionally successful in moving to five defects per hundred thousand lines of code. No software-maintaining enterprise has yet documented bettering this level of freedom from defects.

5.5.3. Domain architecture

To achieve effective maintenance, the enterprise understands that it must obtain a profound knowledge of its application domain and architecture (see Figure 5). Capable domain architecture practices provide the foundation for software reuse. This fundamental knowledge is generally not available in most organizations. What little exists usually is in the minds of the personnel, and not held by the enterprise.

5.5.4. Product traceability

The enterprise establishes and sustains product traceability among life cycle documentation, including requirements, specifications, designs, code and test procedures (see Figure 5). The National Software Quality Experiment described later in this paper revealed that over 44% of the defects detected in software inspections result from lack of product traceability (O’Neill, 1997b).

5.6. Customer contribution

5.6.1. Customer satisfaction

To achieve effective maintenance, the enterprise measures customer satisfaction in the context of its software operations and products, and establishes upper and lower limits for its management and control. Figure 6 summarizes the estimated shortfalls. Software products and features provide economic value to the enterprise and drive improved customer satisfaction.

5.6.2. Deliver value

To achieve effective maintenance, the enterprise assesses the degree to which its products and services deliver value to its customers and the degree to which software is essential (see Figure 6). The enterprise may have software products and processes that are strategically essential to the competitiveness of the business. These products and processes can deliver substantial economic value to the enterprise.

5.6.3. Software failure

To achieve effective maintenance, the enterprise takes active steps to guard against software failure and to restore operations in the event of failure, and provides oversight for their effectiveness (see Figure 6). People make mistakes sometimes. Creating software is labour intensive, and there is an increasing amount of software in every industry sector. In addition, the trustworthy practice of software engineering is immature. Consequently,

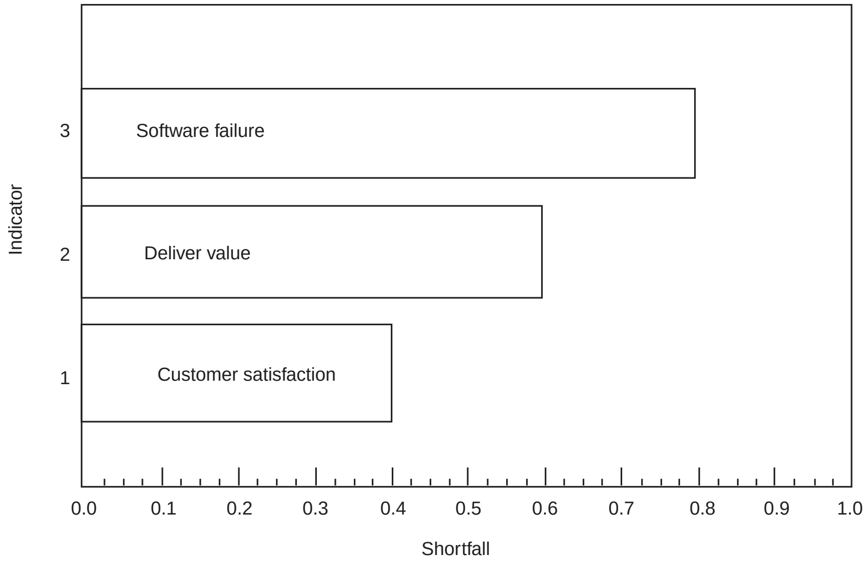

Figure 6. Estimated shortfalls from customer contribution characteristics

the risk of software failure is greater than zero with the current software maintenance methods, techniques, practices, processes and their management.

5.7. Government research and development

To achieve effective maintenance, the enterprise understands the scope of government resources and research and how to obtain their results for the benefit of the enterprise. Figure 7 gives an estimate of the shortfall. As noted previously, government research and development has fallen from from 2·8% to 2·2% of GDP over the past ten years.

6. NATIONAL SOFTWARE QUALITY EXPERIMENT

6.1. Defects detected

The National Software Quality Experiment (NSQE) is a mechanism for obtaining samples of software product quality in order to reveal patterns in a nation's software infrastructure (O'Neill, 1997b). Thousands of participants from dozens of organizations are populating the experiment database with thousands of defects of all types, along with pertinent information needed to make analyses meaningful. The distribution of defects by defect type serves as a risk management vector useful in planning for software maintenance. Table 1 and Figure 8 summarize the defect categories that account for about 92% of all defects detected and reported.

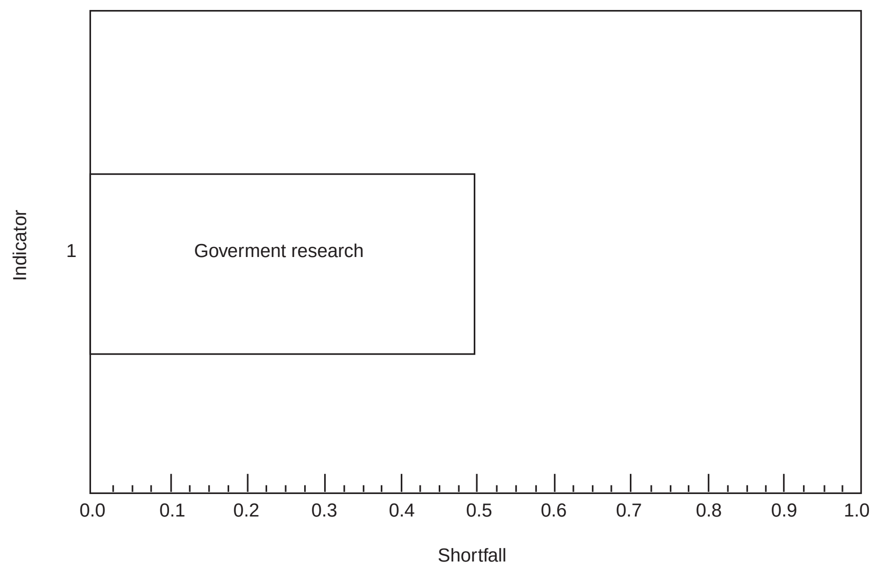

Figure 7. Estimated shortfall from government research and development

Table 1. Defects detected and reported in the NSQE

Category	Defect ratio	Example of defect represented by the category
Documentation	44·27%	Documentation and implementation do not match
Standards use	21·05%	Non-compliance with applicable enterprise standard
Logic	7·65%	Correctness questions revealed error
Functionality	6·36%	Intended function not stated or not met
Data	5·00%	Data definition error; initial value setting error
Syntax	4·66%	Compliance with programming language syntax
Performance	2·59%	Weak management of computer resources in application
Other errors	8·42%	Wide range of miscellaneous errors
Total	100·00%	Total of all defects reported in NSQ experiment

6.2. Risk management aspects

The reported relative frequency of the defects in the seven categories is relevant to managing software risks on software maintenance projects. Here it is useful to make a careful distinction between the sources of risk, the risks and the consequent problems. A consequent problem is a previous risk that actually happened with its consequences still being played out.

The lack of a requirements traceability mechanism is a principal source of risk in the documentation defect category. Left unattended, this source of risk will continue at a high probability of occurrence, and there will be problems in baseline and change management, and in ease of maintenance and adaptability. These documentation defect risks are not regarded as being important by some managers in the field, even though documentation has long been identified as a major problem in doing software maintenance (for example, (Chapin, 1985)).

The lack of an enterprise standard operating practices guide covering software maintenance, or the lack of its enforcement, is a principal source of risk in the standards defect category. Left unattended, this source of risk will continue at a moderately high probability of occurrence, and there will be problems in the ease of maintenance and adaptability as 'coding cowboys' continue to roam off in random directions. These standards risks are not regarded as being important by some managers in the field.

The lack of shared vision between the software producer and consumer, and the lack of experience with the application domain are the principal sources of risk in the functionality defect category. Left unattended, this source of risk will continue at a moderate probability of occurrence, and there will be problems in end-user satisfaction.

The lack of rigour in applying systematic design technology, structured programming and disciplined data structures are the principal sources of risk in the logic defect category. Left unattended, this source of risk will continue at a moderate probability of occurrence, and there will be problems in reliability and ease of maintenance and adaptability.

The lack of experience with the rules of syntax, and the fact that people sometimes

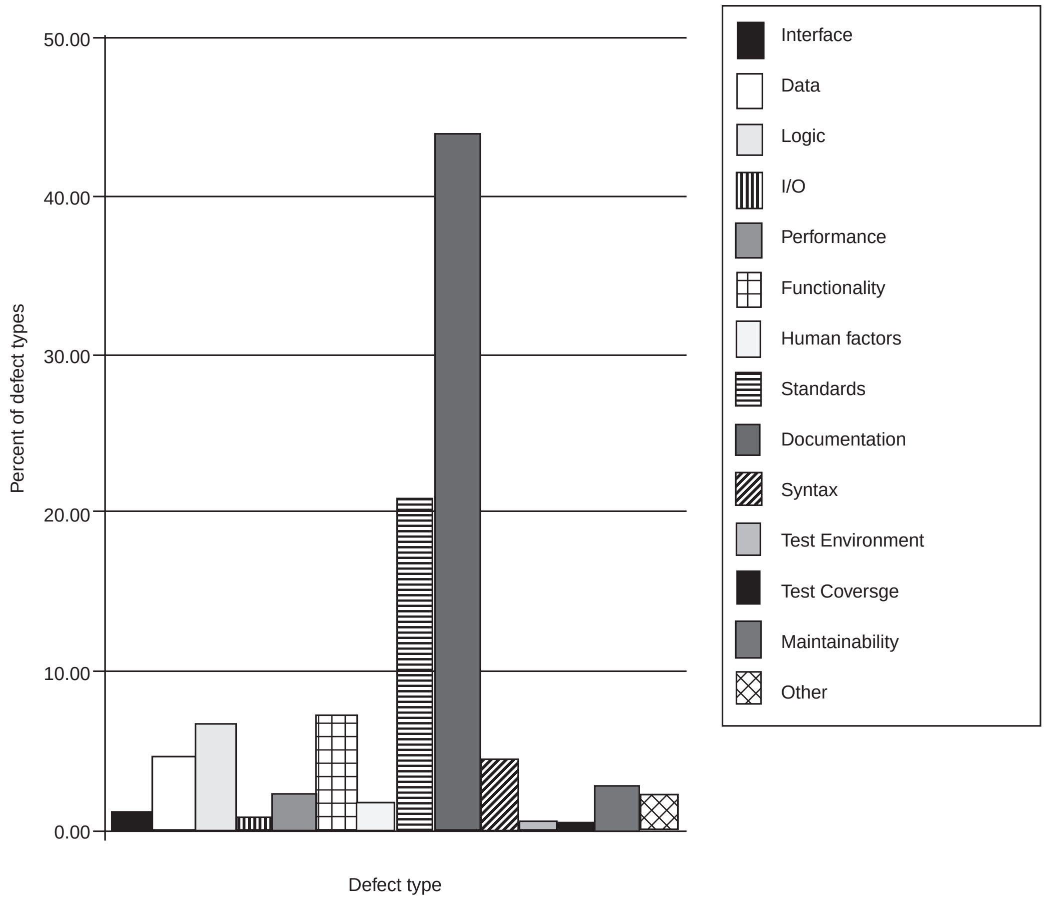

Figure 8. A summary classification of defects for 1992 through 1996 detected and reported in the NSQE

make mistakes are the principal sources of risk in the syntax defect category. Left unattended, this source of risk will lead to ambiguity and errors in logical expression, and problems in end-user satisfaction and software adaptability.

The lack of experience with the application domain, with the use of disciplined data structures and with the hardware/software platform are the principal sources of risk in the data defect category. Left unattended, this source of risk will continue at a low probability of occurrence, and there will be problems in reliability.

The lack of experience with the application domain and the hardware/software platform is a principal source of risk in the performance defect category. Left unattended, this source of risk will continue at a low probability of occurrence, and there will be problems in end-user satisfaction as a result.

7. ACTION POINTS AND DISCUSSION

The first six sections of this paper have basically restated common knowledge. They have summarized what has been learned, known and reconfirmed by many able workers drawing upon decades of experience with software maintenance. But in truth, few managers of software maintenance, and their managers above them, recognize the importance to their enterprises and to their nations of what those six sections summarized. For that reason, this paper is a call to action, to take the opportunity to benefit our enterprises and our nations by improving the way we do software maintenance. Continuing to muddle along opens wider the door for other enterprises and other nations to surpass ours. Assuming we consistently meet our enterprises' needs for effective information systems, the key measure of our success in improving software maintenance is improving our span of responsibility.

With 70% of the software resource devoted to supporting software products, software maintenance costs will dominate the enterprise's consumption of IT (information technology) resources. In contemplating action, the usual critical questions must be addressed:

- What is the benefit–cost picture for the enterprise?
- Who stands to gain, who to lose, and how and why?
- What manager is willing to serve as the champion?
- How does the action fit with the enterprise's long-range plan?

In addition, the enterprise's environment must be considered:

- How is the infrastructure changing?
- What is the competition doing?
- What is the industry doing?
- What is the region doing?
- What are the governmental and other entities doing?

While a wide variety of actions may be necessary or beneficial to an enterprise in improving its global software competitiveness, specific actions should be selected to focus the enterprise's limited resources where they are most needed to facilitate competitiveness through improved software maintenance. To improve the span of responsibility to more than 500 000 lines, those actions are likely to be from among the actions in the categories listed below:

- Establish and use organizational resources to sustain operational support (see Section 3).
- Apply resources to strengthen each of the four views of software maintenance (see Section 4).
- Create and apply action to eliminate the main shortfalls in doing software maintenance (see Section 5).
- Reduce risks to information system quality (see Section 6).

In the USA, a national software policy is needed, one that would impact education, research, security, immigration, tax policy and infrastructure, and co-ordinate key initiatives from government, industry and academia. Government policy often results in overreaching or understated actions to correct situations that have already happened or are imminently inevitable, and which bring with them unintended side-effects. This tendency and the ignorance of the USA's leaders about software and information technology could make a national software policy a two-edged sword. Seven possible components of a national software policy for the USA are:

1. Formulate, establish and implement an effective national software policy.
2. Utilize tax policy to allow current year expensing of software changes that are corrective or perfective.
3. Utilize tax policy to allow current year expensing of software reuse expenditures.
4. Utilize tax policy to accord key employee status to critical software maintenance technical staff members.
5. Fund research and development to improve product software maintenance processes.
6. Fund research and development to improve software reuse.
7. Encourage the creation and use of national and international standards relevant to software maintenance.

Faced with enterprises' increasingly heavy burden of software maintenance, and with software becoming an integral part of the functioning of every industry, software maintenance is moving into position to be a major influence on an enterprise's global competitiveness and, by extension, on national prosperity. At the same time, enterprises elsewhere with competitive wage structures and highly trained personnel resources eagerly await not only a major offshore outsourcing workload from enterprises unable to manage successfully their span of responsibility, but also the opportunity to take the leadership position (O'Neill, 1997a).

8. CONCLUSION

The effective management of software maintenance is one of the keys to enterprise success, since information systems and the use of information technology (IT) are increasingly vital to enterprise health. The enterprise that raises its ability to improve software maintenance as a core competence will improve its competitive position. This paper has offered a call to action.

Today's software maintenance practice evolved to support software systems not designed and implemented for maintenance. The result is a low span of responsibility. Tomorrow's software maintenance practice may determine the global software competitiveness and success of an enterprise, and collectively affect the prosperity of a nation. The successful enterprise will take action to sustain operational support, to strengthen each level or view of software maintenance, to eliminate or mitigate the main shortfalls in doing software maintenance, and to reduce the risks to information system quality. Successfully done, such action can result in significant improvements in the span of responsibility.

To the degree an enterprise allows dependence on uncustomized commercial off-the-

shelf software products, it invites its competition to establish an equivalent software infrastructure and hence, equivalent enterprise capability. The world-class enterprise skillfully mixes use of vendor-modified off-the-shelf commodity software and outsourcing of maintenance, with in-house custom maintenance of add-value custom software to achieve competitive superiority efficiently.

References

Buxton, J. M., Naur, P. and Randell, B. (Eds) (1976) *Software Engineering: Concepts and Techniques*, Mason/Charter Publishers, New York NY, 306 pp.

Chapin, N. (1985) 'Software maintenance: a different view', in *AFIPS Conference Proceedings Volume 54 NCC*, AFIPS Press, Reston VA, pp. 328–331.

Chapin, N. (1988) 'Software maintenance life cycle', in *Proceedings of the Conference on Software Maintenance—1988*, IEEE Computer Society Press, Los Alamitos CA, pp. 6–13.

Drucker, P. F. (1974) *Management: Tasks, Responsibilities, Practices*, Harper & Row Publishers, Inc., New York NY, 839 pp.

Gibbs, W. W. (1994) 'Software's chronic crisis', *Scientific American*, **271**(3), 86–95.

Humphrey, W. S. (1989) *Managing the Software Process*, Addison-Wesley Publishing Company, Inc., Reading MA, 512 pp.

Humphrey, W. S. (1993) 'Software engineering,' in Ralston, A. and Rielly, E. D. (Eds), *Encyclopedia of Computer Science*, Van Nostrand Reinhold, New York NY, pp. 1217–1222.

Humphrey, W. S. (1997) 'Some thoughts on software product assessments,' Crosstalk, **10**(8), 25–28.

IEEE (1993) *IEEE Standard for Software Maintenance*, IEEE Std 1219–1993, The Institute of Electrical and Electronic Engineers, Inc., New York NY, 39 pp.

ITAA (1977) *Help Wanted: the IT Workforce Gap at the Dawn of a New Century*, Information Technology Association of America, Arlington VA, 6 pp.

Jones, T. C. (1997) *The Global Economic Impact of the Year 2000 Software Problem*, http://www.spr.com, Software Productivity Research, Inc., Cambridge MA, Version 5.2, 9 pp.

Kumar, M. P., Das, V. S. R. and Netaji, N. (1996) 'Offshore software maintenance methodology', *Journal of Software Maintenance,* **8**(3), 179–197.

McCabe, T. J. and Watson, A. H. (1994) 'Software complexity', Crosstalk, **7**(12), 5–9.

Nosek, J. T. and Palvia, P. (1990) 'Software maintenance management: changes in the last decade', *Journal of Software Maintenance,* **2**(3), 157–174.

O'Neill, D. (1989) *Software Inspections Course and Lab*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, 527 pp.

O'Neill, D. (1997a) 'Software value add study', *Software Engineering Notes,* **22**(4), 11–12.

O'Neill, D. (1997b) 'National software quality experiment: a lesson in measurement', in *Proceedings of the Quality Week Conference*, San Francisco, Software Research Institute, San Francisco CA, section 6M2, 40 pp, pp. 1–25.

Price Waterhouse (1997) *Executive Summary—1997 Software Business Practices Survey*, http://www.pw.com, Price Waterhouse, New York NY, 4 pp.

Rosen, S. (1993) 'Software', in Ralston, A. and Reilly, E. D. (Eds), *Encyclopedia of Computer Science*, Van Nostrand Reinhold, New York NY, pp. 1214–1216.

Salisbury, A. B. (1997) 'Toward a software product assessment: an attack on the broader software maintenance problem', Crosstalk, **10**(1), 22–24.

Schach, S. R. (1994) 'The economic impact of software reuse on maintenance', *Journal of Software Maintenance*, **6**(4), 185–196.

SEI (1997) *Process Maturity Profile of the Software Community*, SEMA.5.97, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, 27 pp.

Thompson, W. (1992) 'William Thompson', in Kaplan, J. (Ed), *Familiar Quotations*, Little Brown and Co., Boston, MA, p. 504.

Author's biography:

Don O'Neill is a seasoned software engineering manager and technologist, currently serving as an independent senior engineering consultant. Following his 27-year career with IBM's Federal Systems Division, Don completed a three-year residency at Carnegie Mellon University's Software Engineering Institute (SEI) under IBM's Technical Academic Career Program. As an independent consultant, Don conducts defined programmes for managing strategic software improvement. These include implementing an organizational Software Inspections Process, implementing Software Risk Management, conducting the Project Suite Key Process Area Defined Program and conducting Global Software Competitiveness Assessments. Don has served on the Executive Board of the IEEE Software Engineering Technical Committee and as a Distinguished Visitor of the IEEE. He is a founding member of the National Software Council and of the Washington DC Software Process Improvement Network (SPIN).

